# Car Seat Monitoring System

**Muhannad Al Mjnaa, Electrical Engineering**

Project Advisor: Dr. Richardson

April 28th, 2019

Evansville, Indiana

**Acknowledgments**

I would like to thank Dr. Howe for the project idea and for giving me a car seat. I would also like to thank Dr. Richardson for advice over this project. I would also like to thank Jeff Corn for helping me print a 3D box for this project.

## Table of Contents

## List of Figures



## List of Tables

## I. Introduction

Parents sometimes forget their kid in the car when they are in a rush, which is unsafe and may cause the child's death. According to the National Safety Council, 48 children die every year from being left in hot cars in the United States [1]. A car seat monitor is for any parent with children because it provides more safety.

One possible solution is to integrate sensors into a car seat that will determine if a child is present in the seat. Once the vehicle is turned off, the child should be removed from the car seat within an allotted amount of time or an alarm will sound.

## II. Problem Definition

There are similar products on the market that alert parents whenever they leave a child unattended in a car. Most of them are built-in to the car seat and have expensive sensors. For this project, the car seat monitoring system will be removable and can fit any car seat. Moreover, it will use less expensive sensors to be affordable to every parent.

The project will use sensors to know whether the child is in the seat or not when the car is turned off. The sensor then will communicate with a microcontroller, which will send a signal to the car alarm system.

The Car Seat Monitor System must:

- Determine whether the car is on or off by a vibration sensor added to the car seat and be able to adjust the sensor threshold.
- Determine whether the child is in the car seat or not by using a force resistive sensor.
- Sound alarm when the child is left in the car after a specific amount of time that can be adjusted.

## III. Design Approach

### *A. Hardware Design*

Car seat monitor system consists of a microcontroller, a vibration sensor, and a force resistive sensor. Also, a potentiometer, and LEDs are used for user interface. The Raspberry 3 B [2] is used as a microcontroller. The Raspberry Pi 3 B was chosen for this project because of its low cost, small size, and low power usage. The microcontroller can run its own operating system; It has 1GB of RAM. The microcontroller has a 1.2 GHz clock and uses the Arm cortex-A53. This will help to run a python script. The microcontroller has a 40-pin header that could be used as GPIO, I2C, UART, or PWM [2]. The Raspberry Pi 3 B has an external microSD storage to save python scripts. The Raspberry Pi is powered by a 5v battery.

**Figure 1:** Picture of the Raspberry pi 3 microcontroller.

The vibration sensor used is a SW-420 vibration sensor [3]. The vibration sensor used comparator LM393 to detect if there is any vibration beyond the threshold which can be adjusted by the user using the on-board potentiometer. To power the vibration sensor, it can be connected directly to the 3.3v pin on the Raspberry Pi. The output of the vibration sensor is digital switch outputs 0 and 1[3]. The vibration sensor will output a high signal while the car is on by detecting a vibration from that car. The Raspberry Pi will read that signal from the GPIO pin that connected to the vibration sensor.

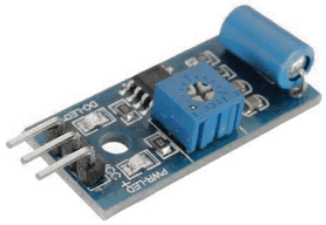

**Figure 2:** Picture of the SW-420 vibration sensor.

In order to detect whether a child is sitting on the car seat or not, a Force Resistive Sensor (FRS) [4] is used. FRS is made of two layers separated by a spacer. FRS is a resistor; the value of the resistance depends on how much it is pressed [4]. FRS ranges from 0 to 20 lbs. However, it will send a signal if the child weighs more than 20 lbs. FRS cannot be connected directly to the Raspberry Pi since the Raspberry Pi does not have an on-board analog to digital convertor (ADC). MCP3008 ADC chip is used to connect the FRS to the Raspberry Pi [5]. The ADC chip has 8 channels. The ADC chip communicates to the Raspberry Pi through a serial peripheral interface bus (SPI). This device only uses two channels (CH0 and CH1).

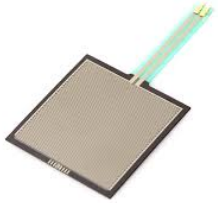

**Figure 3:** Picture Force Resistive Sensor.

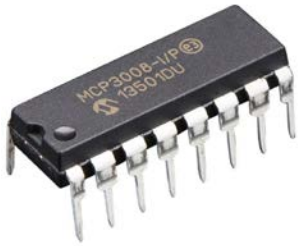

**Figure 4:** Picture of the MCP3008 ADC chip.

For the alarm a 12v horn, a 12v relay and a 12v car battery are used. The car seat monitoring system can be hooked to the built-in car horn, but an external horn and battery are used for demonstration. The horn is 12v, 1.5A, and waterproof. The horn sound output is 105dB, which is enough to alert parents. The DZS Elec 12V relay is used to trigger the horn [6]. The relay has two channels; Only one channel is used for this device. The relay is powered by the 12v car battery and controlled by the microcontroller. The relay trigger current is 5mA and it should be at least 5v.

**Figure 5:** Picture of the 12v Car Horn.

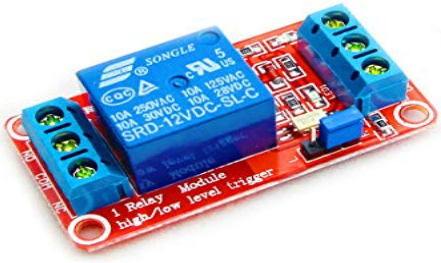


**Figure 6:** Picture of DZS Elec 12V Relay.

The Raspberry Pi GPIO pin output is 3.3v which is not enough to switch the trigger in the relay. A 5v power pin is used to switch the trigger; and to control it, a 2N222 NPN transistor is used [7]. The transistor is used as a level converter and it is controlled by the microcontroller. The microcontroller will send a signal to the alarm system after an amount of time that the user can change using a potentiometer. The user can set the waiting time to 5s, 30s, 60s, 90s, or 120s. 5 LEDs are used to show the selected waiting time.

**Figure 7:** Picture of the 2N2222 NPN Transistor.

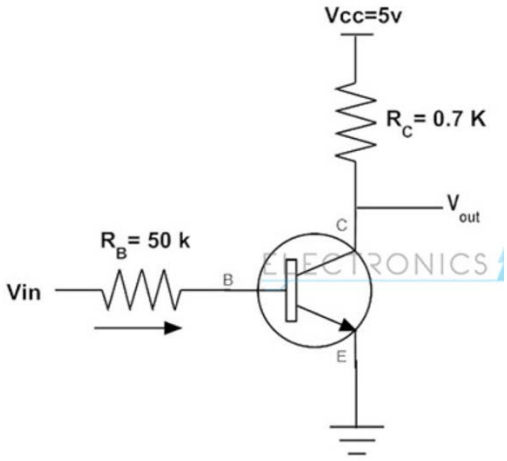


**Figure 8:** Circuit Connection for the Transistor.

The Raspberry Pi, the vibration sensor, the potentiometer, and the LEDs are placed inside a 3D printed box. The box dimension is 7in x 7in x 2in which is big enough to fit these components and small enough to fit any car seat.

A simple circuit connection is shown in figure 9:

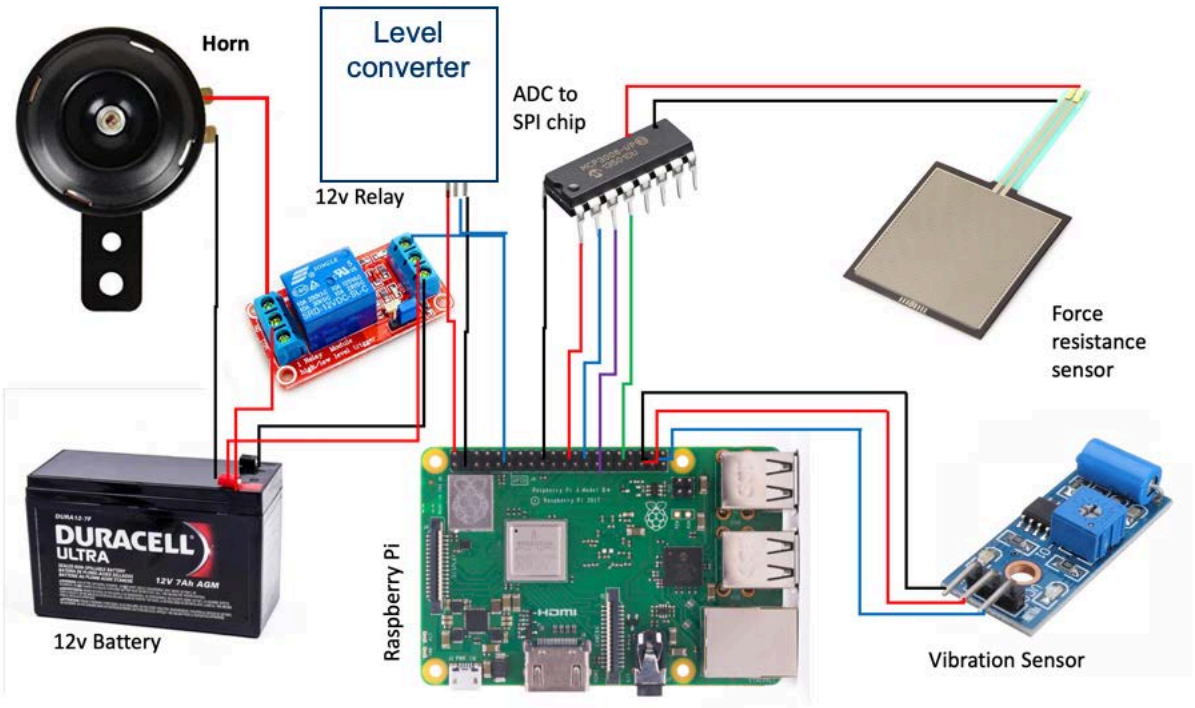


**Figure 9:** Simple Circuit Connection.

*B. Software Design*

Python script is used to run every function in this car seat monitoring system [8]. Python was used because it is easy and has free and open sources. Also, it allows the Raspberry Pi to run automatically without connecting it to a monitor and keyboard. The python code will set up all GPIO pins.

Many libraries have been imported to the code to help setting up components. An Sw40 library has been imported to set up the SW-420 vibration sensor [9]. The library calls different functions based on the state of the vibration sensor. The library also allows the programmer to set up the sensitivity threshold. Another library used in this Python code is Adafruit_MCP3008 [10]; this library is used to set up the MPC3008 ADC chip. This library allows the coder to select the channel to use; this device uses channel 0 for the FRS and channel 1 for the potentiometer. The Python code also uses the time library [11] which allows delay between functions.

The car seat monitoring system has four statuses; the car is on and no child is in the car seat, the car is on and a child is in the car seat, the car is off and no child is in the car seat, and the car is off and a child is in the car seat. Not all statuses will send a signal to the alarm system except when the car is off and a child is in the car seat. The signal will be sent after the waiting time that the user has selected. The alarm will sound after that time until the child is removed, or the car is turned on. The Python code is set to check if the car is on or the child is being removed every second.

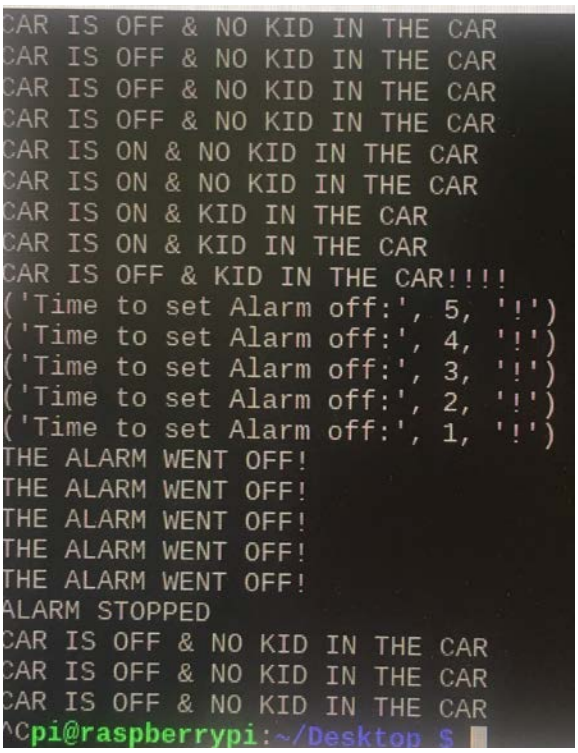


**Figure 10:** Picture of the Raspberry Pi Terminal Updating every second

The system diagram is shown in figure 11:

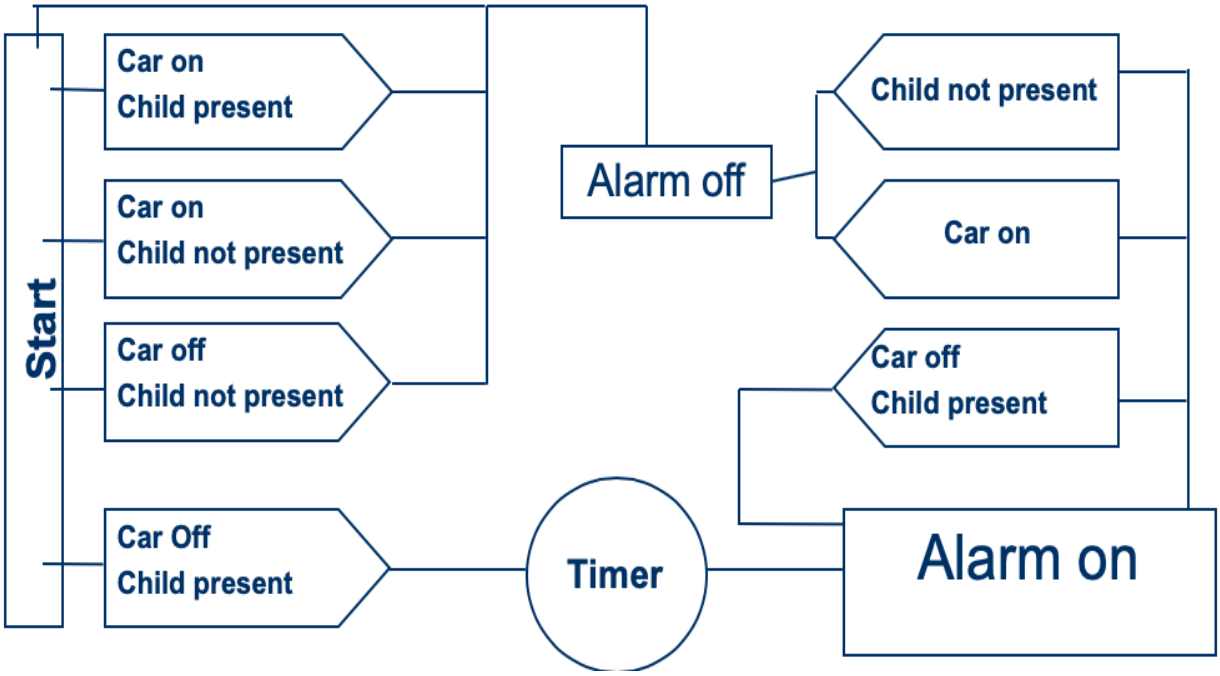


**Figure 11:** System Diagram

*C. Constraint*

Health and safety constraints of this device were considered. There are no additional health and safety risks involved to build or use this device. The device has low voltage power so there are no concerns with it. Plus, the box does not have sharp edges that can harm anyone. Moreover, political constrains were considered. There are no political concerns or limitations to use or build this device.

The economic factor is generally one of the major limiting factors of any design project. To maintain economic feasibility, less expensive sensors are used to build this device. The car seat monitoring system is designed to fit most car seat sizes; therefore, it can be reused. Moreover, a 3D printer is used to build this device, which helps environmentally by using less material to build it.

Manufacturability constraints were also considered. The device has a level of simplicity that can be rebuilt easily. Also, the manufacturer could print a 3D box of a different size and use the same components and codes to build it. Furthermore, sustainability constraints were considered. All components are connected using jump wires, so components can be replaced easily in case they stop working.

The IEEE standard of ISO/ICE/IEEE 24748-5-2017 was considered by using the Raspian operating system on the Raspberry Pi. This standard is met by the team who designed Raspian therefore meets the standard of IEEE [9].

*D. Costs*

The device is relatively inexpensive to be useful for general consumers. Table 1 shows the total cost of this device. Some components are not included in the cost such as resistors and transistors since the university's stock room provides them. The average price of a car seat monitor is $150, proving that this car seat monitoring system is inexpensive to build.

**Table 1:** Car Seat Monitoring System Cost

| Item | Quantity | Cost |
|---|---|---|
| Raspberry Pi 3 B | 1 | $37.30 |
| SW-420 Vibration sensor | 1 | $7.49 |
| Force Resistive Sensor | 1 | $14.99 |
| 12v Horn | 1 | $6.50 |
| DZS Elec 12V Relay | 1 | $8.39 |
| 3D printed box | 1 | NA |
| LED | 5 | NA |
| Resistor | 10 | NA |
| Potentiometer | 1 | NA |
| Battery Holder | 1 | NA |

| | | |
|---|---|---|
| **Total** | - | $74.67 |

## IV. Results and Conclusions

The project meets all standard client requirements. Additional features can be added to this device. One feature is to power up the Raspberry Pi from the car outlet power, so the user will not worry about changing batteries. Another feature that can be added is to use a Bluetooth relay which makes hooking up the device easier. Moreover, connecting a wire to the car outlet will determine whether the car is on or off better than a vibration sensor because newer cars have quiet engines. It is possible to add a cellular board that allows the car seat monitoring system to send a text message to the user's cell phone. Figures 12 and 13 show the final pictures of the product.

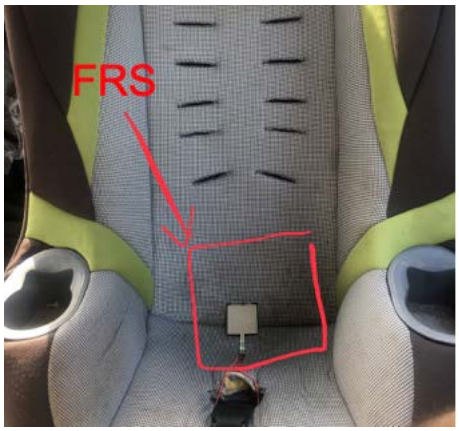


**Figure 12:** Picture of the placement of the FRS.

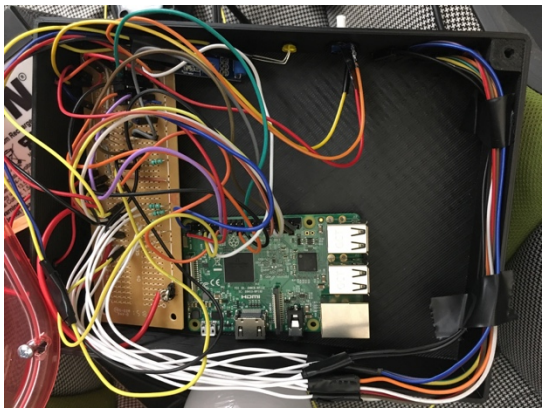

**Figure 13:** Picture of components inside the 3D printed box.

## Appendix A

The Python code for this project:

```
import RPi.GPIO as GPIO
import time
import Adafruit_MCP3008
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
##GPIO.setup(buttonPin1, GPIO.IN, pull_up_down = GPIO.PUD_UP)
##GPIO.setup(buttonPin2, GPIO.IN, pull_up_down = GPIO.PUD_UP)
larm = 21
led1 = 19
led2 = 13
led3 = 6
led4 = 5
led5 = 4
GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
GPIO.setup(led3, GPIO.OUT)
GPIO.setup(led4, GPIO.OUT)
GPIO.setup(led5, GPIO.OUT)
GPIO.setup(larm, GPIO.OUT)
#FRS setups
CLK  = 11
MISO = 9
MOSI = 10
CS   = 22
mcp = Adafruit_MCP3008.MCP3008(clk=CLK, cs=CS, miso=MISO, mosi=MOSI)
values = mcp.read_adc(0)

class Sw40(object):
    def __init__(self, pin , led):
        self.led = led
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.led,GPIO.OUT)
        self.pin = pin
        GPIO.setup(self.pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.add_event_detect(self.pin, GPIO.RISING, callback=self.callback, bouncetime=1)
        self.count = 0

    def callback(self , pin):
        self.count += 1
```

```
    def LedOn1(self):
        carState = True
        GPIO.output(self.led , 1)
        print ("CAR IS ON & KID IN THE CAR")


    def LedOn0(self):
        GPIO.output(self.led , 1)
        print ("CAR IS ON & NO KID IN THE CAR")

    def LedOff1(self):
        GPIO.output(self.led , 0)

    def LedOff0(self):
        GPIO.output(self.led , 0)
        print ("CAR IS OFF & NO KID IN THE CAR")

def main():
    sensor = Sw40(17,26)
    values = mcp.read_adc(0)
    countdown =5 + (mcp.read_adc(1)/1023.0)*115

    try:
        while True:

            countdown =5 + (mcp.read_adc(1)/1023.0)*115
            if countdown <= 6:
                GPIO.output(led1, True)
                GPIO.output(led2, False)
                GPIO.output(led3, False)
                GPIO.output(led4, False)
                GPIO.output(led5, False)
                print('5sec')
                countdown = 5
            elif countdown >6 and countdown <=30:
                print('30sec')
                GPIO.output(led1, True)
                GPIO.output(led2, True)
                GPIO.output(led3, False)
                GPIO.output(led4, False)
                GPIO.output(led5, False)
                countdown = 30
            elif countdown >30 and countdown <=60:
                print('60sec')
                GPIO.output(led1, True)
                GPIO.output(led2, True)
```

```
    GPIO.output(led3, True)
    GPIO.output(led4, False)
    GPIO.output(led5, False)
    countdown = 60
elif countdown >60 and countdown <=90:
    print('90sec')
    GPIO.output(led1, True)
    GPIO.output(led2, True)
    GPIO.output(led3, True)
    GPIO.output(led4, True)
    GPIO.output(led5, False)
    countdown = 90
elif countdown >90 and countdown <=120:
    print('120sec')
    GPIO.output(led1, True)
    GPIO.output(led2, True)
    GPIO.output(led3, True)
    GPIO.output(led4, True)
    GPIO.output(led5, True)
    countdown = 120

time.sleep(1)
GPIO.output(larm, True)

if sensor.count >=10 and mcp.read_adc(0)>500:
    sensor.LedOn1()
    time.sleep(1)
elif sensor.count >=10 and mcp.read_adc(0)<500:
    sensor.LedOn0()
    time.sleep(1)
elif sensor.count <=10 and mcp.read_adc(0)>500:
    sensor.LedOff1()

    print ("CAR IS OFF & KID IN THE CAR!!!!")

    while (countdown > 0) and mcp.read_adc(0)>500 and sensor.count <=10:

        print("Time to set Alarm off:",countdown,"!")

        time.sleep(1)
        countdown -=1
        while (countdown <= 0) and mcp.read_adc(0)>500 and sensor.count <=10:
            GPIO.output(larm, False)
            print("ALAAAARM1111")
            time.sleep(.3)
            GPIO.output(larm, True)
```

```
                print("ALAAAARM000")
                time.sleep(2)
            GPIO.output(larm, True)
        time.sleep(1)
    elif sensor.count <=10 and mcp.read_adc(0)<500:
        sensor.LedOff0()
        time.sleep(1)
    sensor.count = 0

except KeyboardInterrupt:
    GPIO.cleanup()

if __name__ == '__main__':
    main()
```

## References


[1] “Kids and Hot Cars”. National Safety Council. Spring Lake. IL. [Online].
Available: https://www.nsc.org/road-safety/safety-topics/child-passenger-safety/kids-hot-cars

[2] ”Raspberry Pi 3 Specs, Benchmarks & more”
[Online].Available: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[3] “Vibration Sensor Module - SW-420”. [Online].
Available: https://www.elecrow.com/vibration-sensor-module-sw420-p-525.html

[4] “Force Resistance Sensor - Square”. SparkFun Electronics. [Online].
Available: https://www.sparkfun.com/products/9376

[5] “MCP3008 – 8 Channel 10-bit ADC with SPI Interface”
Available: https://www.adafruit.com/product/856

[6] “DZS Elec 12V Relay”. StackExchange.
Available: https://electronics.stackexchange.com

[7] “2N2222” NPN Transistors. Available: http://www.farnell.com

[8] “Python”. Available: https://www.python.org

[9] “IEEE 24748-5-2017 - ISO/IEC/IEEE International Standard” - Systems and Software Engineering--Life Cycle Management--Part 5: Software Development Planning
Available: https://standards.ieee.org/standard/24748-5-2017.html